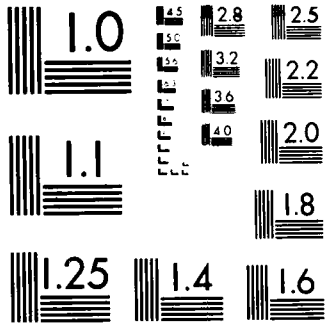
AD-A193 298 HEARTS: A DIALECT OF THE POKER PROGRAMMING ENVIRONMENT SPECIALIZED TO SYSTOLIC COMPUTATION(U) WASHINGTON UNIV SEATTLE L SNYDER OCT 86 TR-86-10-01 N00014-86-K-0264 1/1

UNCLASSIFIED F/G 12/5 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(4)

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: none	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): Hearts: A Dialect of the Poker Programming Environment Specialized to Systolic Computation		5. TYPE OF REPORT & PERIOD COVERED: Technical Report 6. PERFORMING ORG. REPORT NUMBER: 86-10-01
7. AUTHOR(s): Lawrence Snyder		8. CONTRACT OR GRANT NUMBER(s): N00014-86-K-0264
9. PERFORMING ORGANIZATION NAME AND ADDRESS: University of Washington, Seattle, Washington 98195		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, Information Systems Program, Arlington, VA 22217		12. REPORT DATE: October 1986 13. NUMBER OF PAGES: 9
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report): Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

parallel programming, Poker Programming Environment, systolic arrays programming environments, graphical programming

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The design of a parallel programming environment specialized to systolic computation is proposed. The system, called Hearts, is a dialect of the Poker parallel programming environment. The key feature of Hearts that enables it to be a convenient and efficacious facility for writing systolic programs is a novel concept of "program" that is graphical rather than textual. The use of this program form is illustrated in a full example of the Kung-Leiserson matrix product algorithm.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

HEARTS: A DIALECT OF THE POKER PROGRAMMING ENVIRONMENT SPECIALIZED TO SYSTOLIC COMPUTATION

LAWRENCE SNYDER

INTRODUCTION

Because systolic algorithms are commonly thought of as being directly implemented as hardware arrays, writing systolic programs would appear to be an activity without application, and therefore without need for a programming environment. But the appearance is deceiving. There are many times when one indeed does program systolic algorithms: when the systolic array is programmable, during the design process (for simulation purposes) of hardwired array implementations, when a systolic algorithm is used on a general purpose parallel computer, or when one is engaged in research on systolic algorithms. Furthermore, systolic arrays share with other parallel algorithms the characteristic of being deceptively complex – perhaps their simplicity makes the deception even greater – so the benefits of a programming environment to ease the programming task become very important. In this paper we describe the design of a parallel programming environment specialized to systolic computation and illustrate its use.

The "root language" from which the current proposal derives is the Poker parallel programming environment [Snyder, 1984]. Although Poker was originally developed for the CHiP family of computers [Snyder, 1981], including the Pringle [Kapauan, *et al.* 1984], the language has exhibited much wider applicability. Specifically, Poker is being retargetted [Snyder and Socha, 1986] for the Caltech Cosmic Cube [Seitz, 1985] and it has recently been argued [Snyder, 1986] to be ideal as the basis for a systolic programming environment. The design of this new environment, called Hearts, is the topic of the present paper.

One of the key ideas of Poker, inherited by Hearts, is the novel notion of "program". Specifically, the programmer's view of the program appears to be a dynamic version of a textbook illustration. Such metaphorically rich pictures correlate closely with the programmer's thinking and thus simplify the programming task. (The next section is an example illustrating this similarity between the textbook and Hearts descriptions.)

Although Poker was just described as a "language", we also use the name to refer to the whole programming environment. In this sense Poker, and by extension Hearts, is an integrated set of facilities providing full support for editing, compiling, assembling, loading, tracing, debugging, as well as an interface to the underlying UNIX™ operating system, certain correctness checks, the external file system, library and "help" facilities. The system is so complete that the user need not exit until the session is over; moreover, the system is sufficiently well integrated that the user moves effortlessly between the various "subsystems" almost unaware that the change of activity has required different support from the environment.

$$\begin{bmatrix} a_{11} & a_{12} & & & 0 \\ a_{21} & a_{22} & a_{23} & & \\ a_{31} & a_{32} & a_{33} & a_{34} & \\ & a_{42} & & \ddots & \\ 0 & & & & \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} & & 0 \\ b_{21} & b_{22} & b_{23} & b_{24} & \\ & b_{32} & b_{33} & b_{34} & b_{35} \\ & & b_{43} & \ddots & \\ 0 & & & & \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} & 0 \\ c_{21} & c_{22} & c_{23} & c_{24} & \\ c_{31} & c_{32} & c_{33} & c_{34} & \\ c_{41} & c_{42} & & \ddots & \\ 0 & & & & \end{bmatrix}$$

A B C

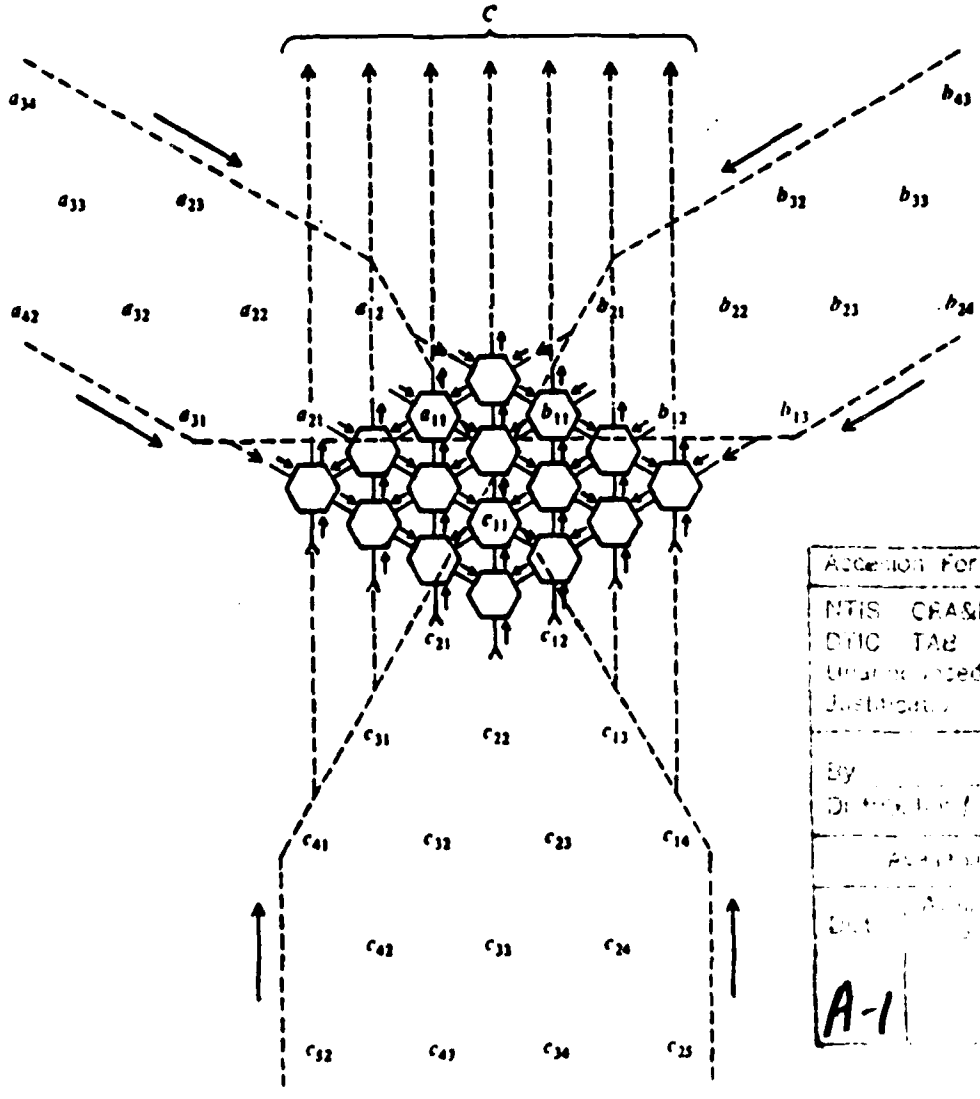


Figure 1

illustrated in the present example, for synchronization reasons. In any case, more than one process may be defined, and processes may have parameters in order to particularize them more easily.

The sequential language used for process definition requires some extensions and some specialized semantics to simplify programming systolic computations. In the declarations, trace variables provide a means of dynamically displaying the computation, and ports declares the names to be used for the processor's datapaths. (Both features are further explained below.) The arrow operator,

$$< variable > \ <- \ < portname >$$
$$< portname > \ <- \ < variable >$$

specifies input or output depending on whether the port name is on the right or left, respectively. The semantics are that a scalar value is transmitted, reading from a port name that does not correspond to a datapath (see Figure 5) yields a 0, and writing to a port name that does not correspond to a datapath is a noop. The command tock is a synchronisation specifier: Control waits at the tock and proceeds only when the global clocking signal is received[1].

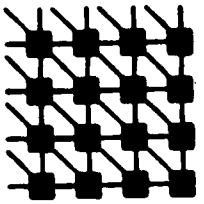


Figure 2

The process code appears to be somewhat more complex than might seem to be needed based on the original Kung and Leiserson description, but what complexity there is can be attributed to declarations and to providing three entry points to the compute-idle-idle cycle. Specifically, the process is parametrized with an integer, *cycle*, specifying whether the process should begin by executing its inner product step (cycle=0), should execute the inner product after one idle step (cycle=1) or after two (cycle=2). A second parameter, *lastval* of type port, states which input stream's termination will signal termination of the process. This specification, permitting values to "drain" out of the array, is required by the fact that, although the end of the *C* array stream generally signals the end of the processing, *C* is actually created internally (the processors on the east and south sides of the array simply read the *Cin* port which returns the default value 0 because (see Figure 6) they do not label an actual data path) and thus will never terminate; the end of the *A* and *B* array streams will terminate processing for processors on the south and east edges of the array, respectively. The choice of terminator for the corner element is arbitrary. The streams are terminated by a special token, EOS, mnemonic for end of stream. The remainder of the code should be self-explanatory. [The branches into the body of the repeat-until loop violate the tenets of software engineering and are used here only to reduce confusion in this presentation; several more acceptable but more verbose solutions are available.]

[1] "Tick" is also used in the system to measure time, but a full explanation goes beyond the scope of this paper.

```
code inner (cycle, lastval);
trace a, b, c;
ports Ain, Bin, Cin, Aout, Bout, Cout;
begin real a, b, c; int cycle; port lastval;

    if cycle = 0 then go to L0          /* Define cycle 0 entry pt*/
    if cycle = 1 then go to L1          /* Define cycle 1 entry pt*/
    a:= b:= c:= 0;                      /* Cycle 2 entry point    */
    repeat
    c:= c + a * b;
    Aout <- a, Bout <- b, Cout <- c;
L2: tock;
L1: tock;
L0: tock;
    a <- Ain, b <- Bin, c <- Cin;
    until EOS(lastval);
    Aout <- a, Bout <- b, Cout <- c;

end.
```

Figure 3

Process Assignment. The process assignment activity associates processes and their actual parameters with processors. The specification is given using a modified version of the communication structure graph in which a window is provided for each vertex. The name of the process to be executed on the processor is entered in the window together with its actual parameters, if any.

The matrix multiplication problem uses only one process having two parameters, *cycle* and *lastval*, although it might have been equally convenient to use three processes – one each of the different positions of the inner product step in the compute-idle-idle cycle – in which case the variable *cycle* would not be required [Snyder, 1986]. Notice that the actual values for *cycle* line up (on counter diagonals); the processors in the main block of the array all get their last value from *Cin*; the choice of *Ain* or *Bin* as the actual value for *lastval* was arbitrary for the corner processor.

Port Name Assignment. Port naming simply labels the datapaths incident to a processor with identifiers used in the process definition. As with the process assignment a modified version of the communication structure graph is used, but for port naming the window associated with each vertex is divided into eight "panes" corresponding to the eight compass points. The port names, although clipped to five characters each in the display, are of arbitrary length.

The specification of the port names for the matrix product program is straightforward; for example, *Bin* is in the northern most pane and *Bout* in the southern most because the *B* matrix flows from the top to bottom. Notice that if a port name labels a direction and there is no datapath incident to the processor at that compass point then reading that port yields a 0 and writing to it is a noop.

Stream Name Assignment. A stream is the sequence of values entering or leaving a systolic array from its perimeter. The purpose of stream name assignment is to specify the direction of data flow and to organize the streams together into logical units by associating like names and indices. These logical units can then be bound to file names, thereby providing the interface between the Hearts system and the underlying file system.

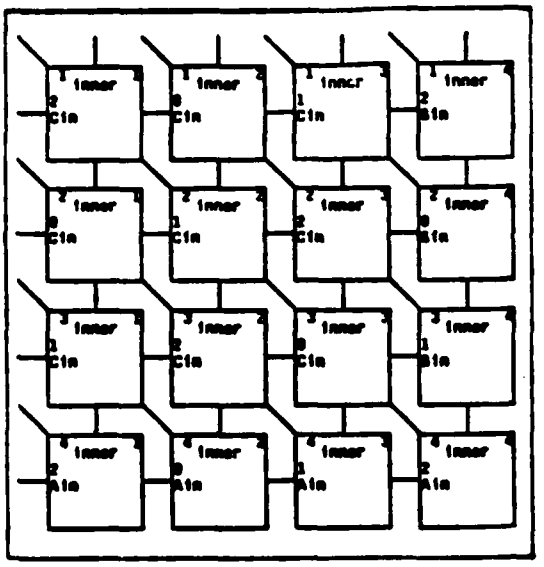


Figure 4

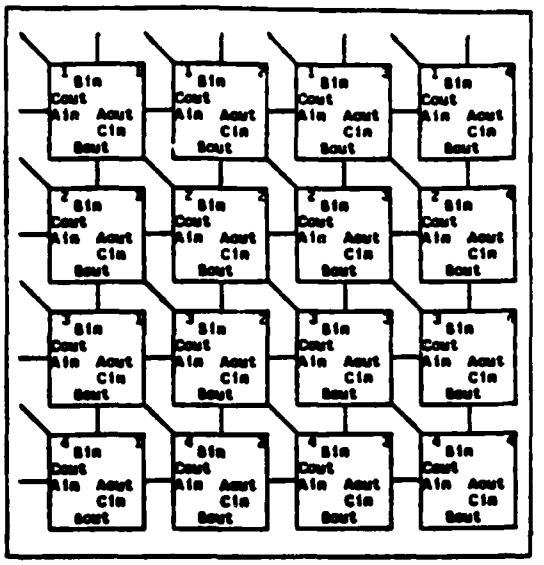


Figure 5

The stream names are given in a table, one entry per "dangling" edge, or *pad*. For each pad a name, a (unique for that name) index and direction of flow are specified. To the right of the vertical line, the table contains copious information derived from the other four constituent parts of the program; this information is provided by the system for its mnemonic value.

The diagonals of the A and B arrays are stream inputs to the program. The fact that each array uses one stream name with indices implies that each array will be stored in its own file, since file names can be bound to stream names. The indices are used to specify the position of the stream within the file: A file with k streams each of n values will contain n fixed length records each with k fields; the index specifies the field position. Thus, the assignment of indices for the A and B array is dictated by how the two band matrices are to be stored in the file. Similarly, the specification of indices for the C array dictates how the streams are to be composed into a file.

Notice that the systolic program is defined in Hearts by three pictures (the communication structure definition, process assignment, and port name assignment), a segment of sequential program text (process definition), and a table (stream name definition); these are not illustrations of the program, they *are* the program. Hence, the Hearts program exhibits the pictorial

STREAM				DESTINATION				
PAD	NAME	INDEX	DIR.	PORT NAME	DIRECTION	CODE NAME	I	J
1	Barray	1	Input	Bin	north	Inner	1	1
2	Carray	5	Output	Cout	northwest	Inner	1	2
3	Barray	2	Input	Bin	north	Inner	1	2
4	Carray	6	Output	Cout	northwest	Inner	1	3
5	Barray	3	Input	Bin	north	Inner	1	3
6	Carray	7	Output	Cout	northwest	Inner	1	4
7	Barray	4	Input	Bin	north	Inner	1	4
8	Aarray	1	Input	Ain	west	Inner	4	1
9	Carray	1	Output	Cout	northwest	Inner	4	1
10	Aarray	2	Input	Ain	west	Inner	3	1
11	Carray	2	Output	Cout	northwest	Inner	3	1
12	Aarray	3	Input	Ain	west	Inner	2	1
13	Carray	3	Output	Cout	northwest	Inner	2	1
14	Aarray	4	Input	Ain	west	Inner	1	1
15	Carray	4	Output	Cout	northwest	Inner	1	1

Figure 6

qualities of the textbook form, though it accomplishes them in a somewhat different way.

With the form of an example program fully established, it is now possible to describe the Hearts programming environment.

THE HEARTS ENVIRONMENT

Hearts is an integrated parallel programming environment using interactive graphics to give the programmer all of the facilities needed to write, debug and execute systolic programs. In this section we describe the main features of the environment.

Hearts, like Poker, does not have a textual form for its programs[2], but rather represents them as a relational database and presents the information to the programmer in a form called a *view*. The benefit of this nontextual form is simple: The programmer need not go to the trouble of encoding the program and the system need not go to the trouble of decoding (parsing) it. The view, prepared by the system at the programmer's direction, provides an interface between human and machine across which the information about the runtime behavior of a systolic array is established without a textual encoding of that information.

Two displays are used to maximize the information available to the programmer: A primary, bitmapped graphics display shows most of the graphical views; the secondary display is used only for process definitions which the programmer creates and modifies using a standard editor. Figure 7 shows a typical view on the primary display. In addition to the four constituent parts of a program, (besides the process definition) that are shown as graphical views, there are other views to support other facilities of the environment. The views are:

Interconnection View: Displays the communication structure of the systolic array; the programmer interconnects the processors by drawing lines with cursor keys or a mouse; see Figure 2. [*Corresponds to Switch Setting View in Poker*].

Code Names View: Displays the process assignment information using an abstraction derived from the communication structure where there is a window for each vertex; the programmer moves from window to window entering the process name and any actual parameters to the process; see Figure 4.

Port Names View: Displays the port name assignment information using an abstraction based on the communication structure similar to the Code Name View; the programmer uses the cursor keys or mouse to move to the various windows and within a window to move to the different panes when the port name is entered; see Figure 5.

I/O Names View: Displays the stream name assignment information using a table, the right half of which is prepared by the system; the programmer moves from line to line, entering the names and indices of the streams and whether they are input or output streams; see Figure 6.

Command Request View: Provides the programmer with the compilation, assembly, linking, and loading functions so the program can be prepared for execution; to emphasize the nonstandard nature of Hearts, one of the available facilities "compiles" the communication graph. Program execution can be initiated here in "production" mode; execution for debugging programs is initiated in trace view.

[2]The process definition is textual, of course, but this is only one of the five program constituents.

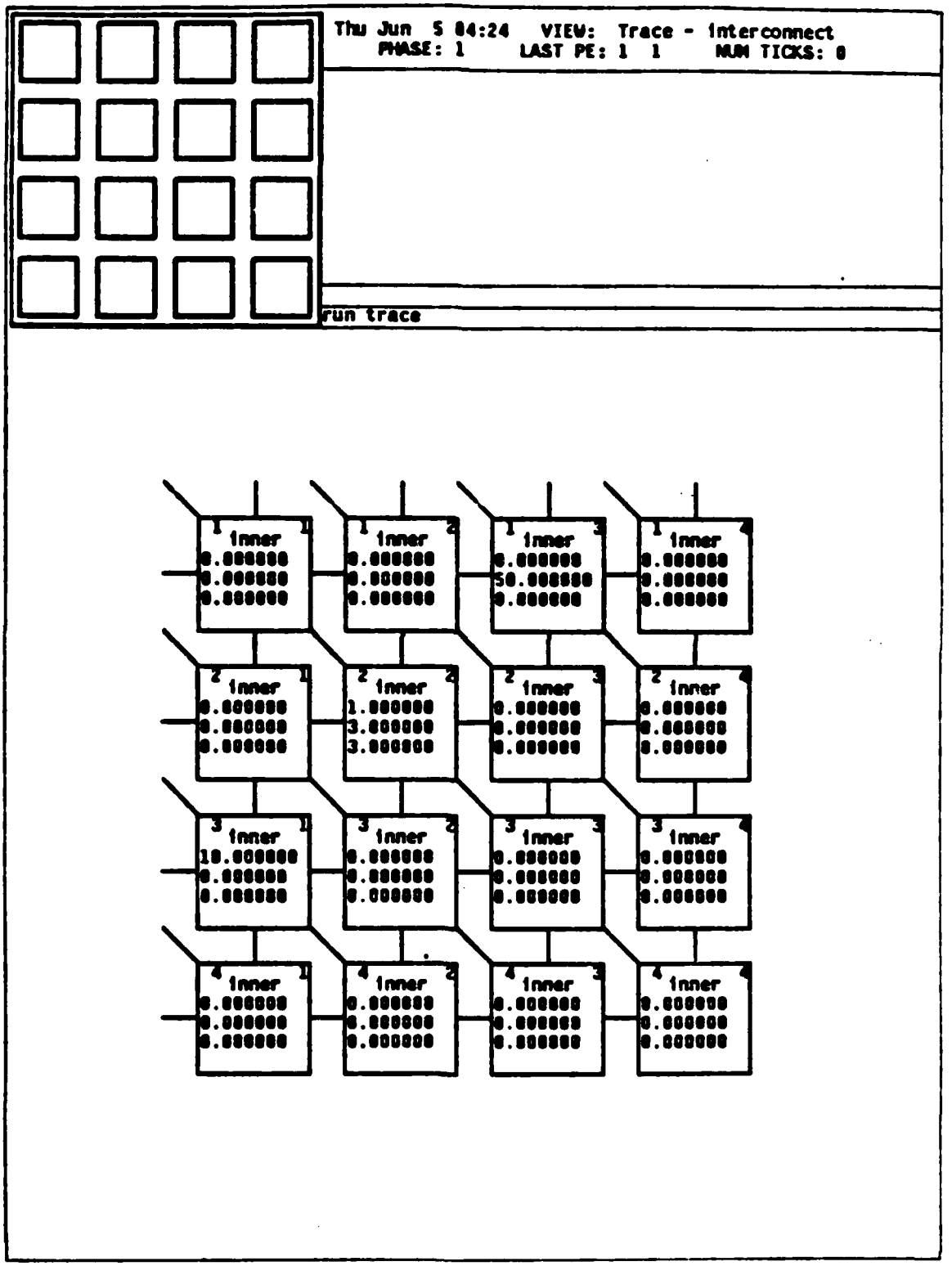

Thu Jun 5 04:24 VIEW: Trace - interconnect
PHASE: 1 LAST PE: 1 1 NUM TICKS: 0
run trace
1 Inner 1
0.000000
0.000000
0.000000
1 Inner 2
0.000000
0.000000
0.000000
1 Inner 3
0.000000
50.000000
0.000000
1 Inner 4
0.000000
0.000000
0.000000
2 Inner 1
0.000000
0.000000
0.000000
2 Inner 2
1.000000
3.000000
3.000000
2 Inner 3
0.000000
0.000000
0.000000
2 Inner 4
0.000000
0.000000
0.000000
3 Inner 1
10.000000
0.000000
0.000000
3 Inner 2
0.000000
0.000000
0.000000
3 Inner 3
0.000000
0.000000
0.000000
3 Inner 4
0.000000
0.000000
0.000000
4 Inner 1
0.000000
0.000000
0.000000
4 Inner 2
0.000000
0.000000
0.000000
4 Inner 3
0.000000
0.000000
0.000000
4 Inner 4
0.000000
0.000000
0.000000

Figure 7

Trace View: Displays the execution of the program while tracing the values of the variables mentioned in the preamble to the process definition; like Code and Port Names Views, Trace uses the abstracted version of the communication graph; see Figure 7 where the a, b, and c values are being traced and execution has been captured at the moment when processor 2, 2 computes its first intermediate result.

System Parameters View: Displays the parameters, e.g. the number of processors, of the array being programmed, and provides the programmer with the ability to change them. [Corresponds to CHiP Parameters in Poker].

Each view provides many other facilities in addition to those just mentioned including screen management facilities, help facilities, access to the underlying operating system, diagnostic facilities, and so forth.

A new Hearts programming session would begin by defining System Parameters to specify the size and type of array to be programmed, as well as other system characteristics. Next the programmer will enter one of the views and begin specifying the information required for that view. Although there is no mandatory order in which the user visits the views, there are some weak dependencies: a program must be defined before tracing is possible, and pads must be defined in the communication structure before stream names can be defined. Generally, the programmer begins with the Interconnection View to define the communication structure, but then moves between views frequently in what may appear to the observer to be a rather random way.

A common property of systolic arrays is that much of the information is repetitious, and so it would appear to be rather tedious to have to specify all of this repeated information. In fact, there are a variety of facilities to enable the programmer to enter repeated information easily. In the Code Names and Port Names views, for example, the same entry can be assigned to all elements of multiple rows or columns with half a dozen key strokes. Similarly, in the I/O Names View it is possible to assign stream names with consecutive indices to multiple pads occurring in a pattern; thus for the Stream Names Definition in Figure 6 the programmer had to make only three complete entries, one for each array, and three repetition commands.

When the program is complete the programmer can either check it by running predicate checks which verifies that the program has certain properties, e.g. all processors are connected in the communication structure, or he can move directly to the Command Request View to compile it. After the program has been successfully compiled and linked it can either be downloaded into the physical hardware, or down loaded into a simulator. In the latter case the programmer can use the Trace View to watch a continuously updated display of the progress of the simulated computation, and thereby observe bugs in the program. Should bugs be found they can be corrected by returning to the appropriate "source view."

CONCLUSIONS

There are two key points about the Hearts parallel programming environment: the novel program structure and the new nontextual style of specifying it. These two features combine to yield a convenient, efficacious programming environment for writing and running systolic programs. In particular, the metaphorically rich pictures that the system provides the programmer offer a perspicuity comparable to dynamic versions of textbook illustrations.

The Hearts system described here is only a design which has not yet been fully implemented. The Poker environment of which Hearts is a dialect, has been fully implemented and has been used to write systolic programs. Not only has Poker provided the concepts and experience needed to design Hearts, (and the figures shown in this paper, too), it represents

about 90% of the facilities of the Hearts implementation. The major differences involve the Interconnection View where the communication structure in Hearts is somewhat more easily specified, the process definition language where more constructs specialized to systolic arrays are provided, and in the backend simulator where a synchronous execution mode must be provided; minor differences abound and reflect improvements derived from our experience with this style of programming.

REFERENCES

Cuny, Janice and Snyder, Lawrence, Compilation of Data-driven Programs for Synchronous Execution, *Proceedings of the 10th Symposium on the Principles of Programming Languages*, ACM, 1983, 197-202.

Kapauan, Alejandro, Field, J. Timothy, Gannon, Dennis and Snyder, Lawrence, The Pringle Parallel Computer, *Proceedings of the 11th International Symposium on Computer Architecture*, IEEE, 1984, 12-20.

Kung, H.T. and Leiserson, C.E., Algorithms for VLSI Processor Arrays. In Mead, Carver and Conway Lynn, *Introduction to VLSI Design Systems*. Addison-Wesley, 1980.

Seitz, Charles L., Cosmic Cube, *CACM* 28(1), 1985, 22-33.

Snyder, Lawrence, Overview of the CHiP Computer, in John P. Gray (editor), *VLSI 81*, Academic Press, 1981, 237-246.

Snyder, Lawrence, Parallel Programming and the Poker Programming Environment, *Computer* 17(7), July 1984, 27-36.

Snyder, Lawrence, A Syntax-free Language for Parallel Computing, University of Washington, 1985.

Snyder, Lawrence, Programming Environments for Systolic Arrays, *First Annual Symposium on Optoelectronics and Laser Applications in Science and Engineering*, S.P.I.E., January 1986.

Snyder, Lawrence and Socha, David, Poker on the Cosmic Cube: The First Retargetable Parallel Programming Language and Environment, *Proceedings of the International Conference on Parallel Processing*, 1986 (to appear).

END

DATE

FILMED

6-1988

DTIC